C-Crashkurs

Literatur:

- B. Kernighan, D. Ritchie: *The C-programming language*, Second Edition, Prentice-Hall, 1988.
- R. Sedgewick: *Algorithms in C*, Third edition, Addison-Wesley, 1998.
- Unix man-Pages

RWTH RHEINISCH-WESTFÄLISCHE TECHNISCHE HOCHSCHULE AACHEN
LEHRSTUHL INFORMATIK XI SOFTWARE FÜR EINGEBETTETE SYSTEME

Inhaltsübersicht

Historie

Weiterentwicklung von BCPL und B

- BCPL: Ende der 60er Jahre von Martin Richards zum Bau von Betriebssystemen und Compilern entwickelt
- B: Ken Thompson erstellte 1970 mit B das erste UNIX System

C

- 1972 von Dennis Ritchie in den Bell Laboratories entwickelt
- Wurde zur Entwicklung des UNIX-Betriebssystems verwendet
- Zunächst durch den Klassiker „The C Programming Language“ von Brian Kernighan und Dennis Ritchie beschrieben und 1989 vom amerikanischen ANSI-Institut standardisiert
- Häufig nicht als Hochsprache angesehen, da maschinennahe Programmierung möglich
- Hohe Flexibilität, kleiner Sprachumfang (ANSI-C hat nur 32 Schlüsselwörter)
- keine Schutzmechanismem, kein strenges Typkonzept
- Programmiersprache für Programmierer

RWTH RHEINISCH-WESTFÄLISCHE TECHNISCHE HOCHSCHULE AACHEN
LEHRSTUHL INFORMATIK XI SOFTWARE FÜR EINGEBETTETE SYSTEME

Ein einfaches Programm

```
#include <stdio.h>                      // Funktionsdeklarationen zur Ein-/Ausgabe einbinden

int maxi( int a, int b );               // Funktionsdeklaration

int global_max = 0;                     // Globale Variable global_max (mit 0 initialisiert)

/*******************************
 * Hauptprogramm
 *******************************/
int main( int argc, char *argv[] ) {    // Hier beginnt die Programmausführung
    printf( "Hello World.\n" );         // Ausgabe von "Hello World." auf der Standardausgabe
    global_max = maxi( 1,2 );           // Aufruf der Funktion maxi
}

/*******************************
 * Funktion zur Maximumberechnung
 *******************************/
int maxi( int a, int b ) {              // Funktionsdefinition
    if (a>b)
        return a;
    else
        return b;
}
```

Datentypen

Elementare Typen

Typ	*übliche* Größe	*minimaler* Wertebereich	Beispiele
char	8 Bit	-127 ... 127	'a', '&', ' '
unsigned char		0 ... 255	'\n','\0'
int	16 Bit	-32767 ... 32767	Dezimal: 0,-7,42
unsigned int		0 ... 65535	
(unsigned) short int	16 Bit	wie int	Oktal: -07,010
long int	32 Bit	-2147483647 ... 2147483647	
unsigned long int		0 ... 4294967295	Hexadez.: 0x2f, 0xa
float	32 Bit	auf 6 Stellen genau	0.0, -1.7
double	64 Bit	auf 10 Stellen genau	31415e-4
long double	128 Bit	auf 10 Stellen genau	

Variablendefinition

```
char c = '7';            // Vorzeichenbehaftete Character-Variable namens c mit Initialisierung
unsigned long int ul;    // Vorzeichenlose 64 Bit Variable namens ul
```


Datentypen – Verbunde, Arrays

Verbunde

```
struct [Verbundname] {
        Datentyp VariablenName;
        Datentyp VariablenName;
        ...
} [Variablenliste];
```

Beispiel

```
struct punkt {
        int x;
        int y;
};
```

Variablendefinition

```
struct punkt p;

struct {
        struct punkt obenLinks;
        struct punkt untenRechts;
} r1, r2;
```

Zugriff

```
p.x = 7;
r1.obenLinks.y = 42;
```

Arrays

Datentyp Varname**[**Anzahl der Elemente**];**

- Arrays in C sind immer 0-basiert: **int a[n]** definiert die Elemente a[0], ..., a[n-1].
- Die Elemente eines Arrays befinden sich in einem zusammenhängenden Speicherbereich, d.h. sie befinden sich an aufeinanderfolgenden Adressen.

Definition

```
char c[26];
float noten[4] = {1.0, 2.0, 3.0, 4.0};
int tabelle [2][5] = {{1,2,3,4,5}, {9,8,7,6,5}};
```

Zugriff auf Elemente

```
c[0] = 'a';
c[25] = 'z';
tabelle[0][4] = 42;
```

Datentypen – enum, typedef

enum

Integer-Variablen, die nicht jeden beliebigen Wert annehmen dürfen, sondern auf eine begrenzte Anzahl von Werten beschränkt sind. Diese Werte werden über Namen angesprochen.

enum [Name] {Element_1, …, Element_N} [Variablenliste];

Beispiel

```
enum Wochentag {So,Mo,Di,Mit,Do,Fr,Sa};
enum Richtung {
  NORD, OST=90, SUED=180, WEST=270};

enum Wochentag heute;
heute = Di;
```

typedef

Ein bereits bestehender Datentyp wird über einen neuen Namen angesprochen.

typedef Datentyp NeuerName;

Beispiel

```
typedef enum Wochentag wtag;
wtag morgen;
morgen = Mi;

typedef unsigned short BOOL;
BOOL bed = (1>0);

typedef struct {
        char name[80];
        int alter;
} tPerson;
tPerson ralf;
ralf.alter = 27;
```


Datentypen - Zeiger

Ein Zeiger ist eine Variable, die eine Speicheradresse beinhaltet. Die Größe und das Format einer Adresse ist BS-abhängig.

Definition

Datentyp *Variablenname;

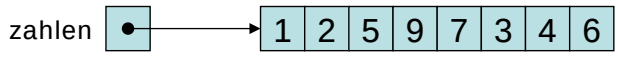
Beispiele

```
int i = 7;
int *p_i;                   // int-Zeiger
int *p_j=NULL;              // Zeiger mit NULL initialisiert
struct punkt *p_punkt;      // Zeiger auf eine Verbundvariable

p_i = &i;                   // & ist Adressoperator: p_i speichert Adresse von i
p_j = p_i;
*p_j = *p_j + 1;            // Dereferenzierung ➔ i hat nun den Wert 8!

(*p_punkt).x = 9;           // Beide Anweisungen
p_punkt->x = 9;             // bewirken das gleiche
```

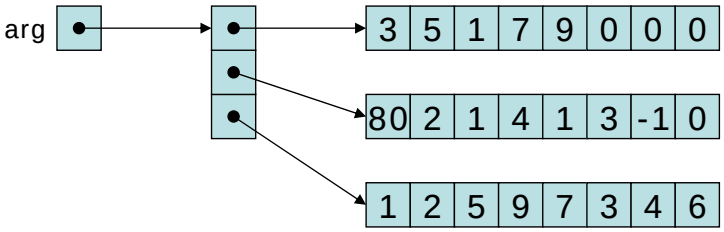
Datentypen - Zeiger und Arrays

Eine Array-Variable ist nichts anderes als ein Zeiger auf das erste Element:

```
int zahlen[8] = {1,2,5,9,7,3,4,6};
```




```
int arg[3][8];
```




```
int *p = zahlen;
*p == p[0];
*(p+1) == p[1];

int **q = arg;              // Zeiger auf einen Zeiger
p = q[1];
```


Operatoren

Arithmetische Operatoren

+	Addition
-	Subtraktion, Negation
*	Multiplikation
/	Division
%	Divisionsrest

Bit-Operatoren

&	Bitweises UND
|	Bitweises ODER
^	Bitweises XOR
~	Invertieren

Shift-Operatoren

<<	Links-Shift
>>	Rechts-Shift

Zeiger-Operatoren

&	Adressoperator
*	Dereferenzierung

Vergleichsoperatoren

==	Gleich
!=	Ungleich
<	Kleiner
<=	Kleiner oder gleich
>	Größer
>=	Größer oder gleich

Logische Verknüpfungen

&&	Logisches UND
||	Logisches ODER
!	Negation

C kennt keinen eigenen Datentyp für Boolesche Variable. FALSE wird durch 0 und TRUE durch einen Wert ungleich 0 repräsentiert.

Typkonvertierungen

Implizite Konvertierung

Enthält ein Ausdruck Variablen oder Konstanten verschiedener Datentypen, wird der Typ einzelner Operanden automatisch umgewandelt. Dabei wird der Operand mit kleinerem Wertebereich an den größeren Datentyp angepasst.

Explizite Konvertierung

Der Programmierer gibt durch Voranstellen des Datentyps an, in welchen Typ die nachfolgende Variable oder Konstante umgewandelt werden soll.

Beispiele

- float f = 1.0/3
  ergibt 0.333333: erster Operand ist Gleitkommazahl, d.h. implizite Konvertierung von 3 zu 3.0 und Durchführung einer Fließkommadivision
- float f = 1/3
  ergibt 0.0: ganzzahlige Division von 1 durch 3 ergibt 0, implizite Konvertierung von 0 zu 0.0
- float f = (int)0.333333 * 3
  ergibt 0.0: explizite Konvertierung von 0.333333 zu 0, anschließende Multiplikation mit 3 und impliziter Konvertierung zu 0.0
- float f = (float)1/3
  ergibt 0.333333: explizite Konvertierung von 1 zu 1.0, implizite Konvertierung von 3 zu 3.0 und Durchführung einer Fließkommadivision
- float f = (float)(1/3)
  ergibt 0.0: ganzzahlige Division von 1 durch 3 ergibt 0, explizite Konvertierung von 0 zu 0.0

Kontrollstrukturen

if - Anweisungen

```
if (Bedingung)
        Anweisung;
[else
        Anweisung;]
```

Beispiele

```
if (a > b)
        max = a;
else
        max = b;
```

```
if (arg[0]=='-')
        if  (arg[1]=='v') {
              version=1;
              printf( "Version 1\n" );
        } else
              printf( "Unknown option.\n" );
```

Verschachtelte if-Anweisungen

```
if (Bedingung1)
        Anweisung1;
else if (Bedingung2)
        Anweisung2;
else if (Bedingung3)
        Anweisung3;
else
        Anweisung4;
```

Bedingungen 1-3 werden in dieser Reihenfolge geprüft. Sobald eine Bedingung erfüllt ist, wird die entsprechende Anweisung ausgeführt. Ist keine Bedingung erfüllt, wird Anweisung4 ausgeführt.

Kontrollstrukturen

switch - Anweisung

Auswahl unter mehreren Alternativen.

```
switch (Ausdruck) {
    case konstanter Ausdruck1:
        Anweisung1a;
        Anweisung1b;
        ...
    case konstanterAusdruck2:
        Anweisung2a;
        Anweisung2b;
        ...
    ...
    case konstanterAusdruckN:
        AnweisungNa;
        AnweisungNb
        ...
    [default:
        AnweisungDa;
        AnweisungDb;
        ...]
}
```

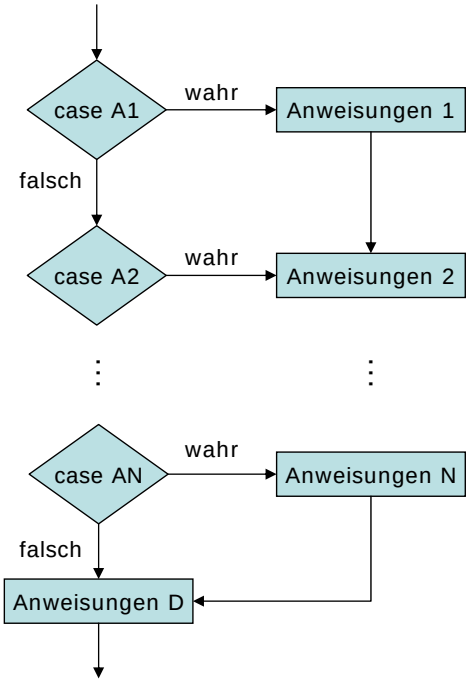
Es wird zunächst der Ausdruck nach switch ausgewertet und das Ergebnis mit allen case-Konstanten verglichen. Stimmt der Wert mit einer Konstanten überein, wird die Programmausführung dort bis zur nächsten break-Anweisung fortgeführt. Ist keine Übereinstimmung zu finden, wird zur (optionalen) default-Marke gesprungen. Ist diese nicht vorhanden, werden keine Anweisungen ausgeführt.

Beispiel

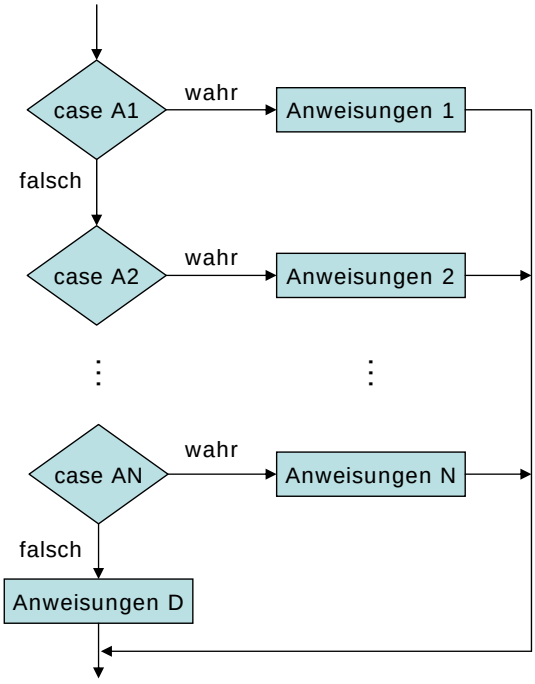
```
char buchstabe;
...
switch ( buchstabe ) {
    case 'a': printf( "a\n" );
    case 'b': printf( "b\n" ); break;
    case 'c': printf( "c\n" ); break;
    default: printf( "Nicht a,b,c\n" );
}
```


Kontrollstrukturen

switch ohne break-Anweisungen



switch mit break-Anweisung



Kontrollstrukturen

while-Schleife

while (Bedingung**)**
    Anweisung;

Solange die Auswertung der Bedingung einen Wert ungleich 0 liefert, wird der Anweisungsblock durchlaufen.

do-while-Schleife

do
    Anweisung;
while (Bedingung**);**

Die Anweisung wird immer mindestens ein mal ausgeführt. Liefert beim Erreichen des while-Statements die Auswertung der Bedingung einen Wert ungleich 0, wird die Anweisung erneut ausgeführt.

for-Schleife

for (AnwInit;Bedingung;AnwSchleife**)**
    Anweisung;

Beim Erreichen des for-Statements wird zunächst AnwInit ausgeführt. Ist danach die Schleifenbedingung erfüllt, wird der Schleifenkörper durchlaufen. Nach jedem Schleifendurchlauf wird AnwSchleife ausgeführt und falls die Schleifenbedingung noch erfüllt ist, ein weiterer Durchlauf gestartet.

Beispiel

```
int i, sum=0;

for (i=0; i<=10; i=i+1)
    sum = sum+i;

for (i=10; i; i=i-1)
    sum = sum+i;
```


RWTH RHEINISCH-WESTFÄLISCHE TECHNISCHE HOCHSCHULE AACHEN
LEHRSTUHL INFORMATIK XI SOFTWARE FÜR EINGEBETTETE SYSTEME

Kontrollstrukturen

break

- Anwendbar bei **switch**, **for**-, **while**- und **do-while**-Scheifen
- Verhindert die Ausführung weiterer Anweisungen innerhalb der Schleife
- Führt zum sofortigen Verlassen von for-, while und while-do-Schleifen
- Kann die Lesbarkeit von Programmen erhöhen, da nicht sämtliche Bedingungen in den Schleifenkopf untergebracht werden müssen

Beispiel

```
while (1) {              // Endlosschleife
    ...
    if ( input==0 )
        break;           // Beenden der Schleife
    ...
}
```

continue

- Anwendbar bei **for**-, **while**-, **do-while**-Schleifen
- Bewirkt eine sofortige Rückkehr zur Schleifenanweisung

Beispiel

```
for (i=0; i<7; i++) {
    if ((i==2) || (i==4))
        continue;
    printf( "%i " );
}
```

Ergibt:
0 1 3 5 6

Dynamische Speicherverwaltung

Häufig ist die Anzahl der zu verwaltenden Datenelemente zur Compile-Zeit unbekannt. Die Verwendung von statischen Arrays würde zu einer ineffizienten Speichernutzung führen.

void *malloc(size_t size)

Reserviert einen zusammenhängenden Speicherblock der Größe size und gibt einen Zeiger auf den Anfang des Blocks zurück. Der Inhalt des Blocks ist undefiniert.

free(void *ptr)

Gibt einen durch malloc() angelegten Speicherblock wieder frei.

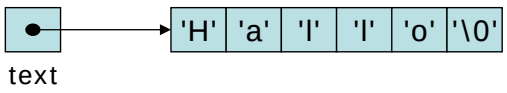
Beispiel

```
char *str;
str = (char *)malloc( 256*sizeof(char) );      // legt einen Speicherblock für 256 char-Werte an
if (str==NULL) {
      printf( "Nicht genügend Speicher.\n" );
      exit(1);
}
...
free( str );                                   // Freigabe des nicht mehr benötigten Speichers
```


Zeichenketten

Zeichenketten werden in C durch Zeiger auf nullterminierte char-Arrays dargestellt.

```
char *text = "Hallo";

char text[6] = {'H','a','l','l','o','\0'};
```



Standardfunktionen zum Manipulieren von Zeichenketten

strcpy, strncpy: Kopieren von Zeichenketten
strcat: Aneinanderhängen von Zeichenketten
strcmp: Vergleichen von Zeichenketten
strchr, strrchr: Suche eines Zeichen innerhalb einer Zeichenketten
strstr: Suche eines Teilstrings innerhalb einer Zeichenkette
strlen: Länge einer Zeichenkette

Funktionen - Definition

Definition

Rückgabetyp Funktionsname**(** Parameterliste **) {**

Anweisungen

}

return (Rückgabewert): führt zum sofortigen Beenden der Funktion und der Rückgabewert wird als Ergebnis an die aufrufende Funktion zurückgegeben.

Beispiel

```
int max;
int maxi (int a, int b) {
        int max=a;              // lokale Variable, die nur innerhalb dieser Funktion gültig ist
        if (a<b)
                max=b;
        return max;             // Verlassen der Funktion und Rückgabe von max
}

void printHallo( void ) {       // Funktion ohne Parameter und Rückgabewert
        printf( "Hallo.\n" );
}

max=maxi( 1,2 );                // Aufruf von maxi mit Parametern 1 und 2
printHallo();                   // Aufruf von printHallo
```


Funktionen - Parameterübergabe

Parameterübergabe ist in C immer Call-by-Value.

Call-by-Value

Änderungen der Parameter innerhalb der Funktion werden außerhalb der Funktion nicht sichtbar.

Beispiel

```
int in;

void func ( int in ) {
        in = in + 1;
}

in=7;
func ( in );        // Nach Aufruf: in==7
```

Call-by-Reference

Wird durch einen "Trick" erreicht: Die Funktion erwartet einen Zeiger auf die Variable.

Beispiel

```
int in;

void func ( int *in ) {
        *in = *in+1;
}

in = 7;
func ( &in );       // Nach Aufruf: in==8
```

Ein-/Ausgabe

printf(Kontrollstring, arg1, arg2, ...)

- Flexible Ausgabefunktion
- Die Funktion wertet den Kontrollstring aus und formatiert die Argumente entsprechend
- Einfacher Text im Kontrollstring wird unverändert ausgegeben
- Platzhalter im Kontrollstring werden durch die Werte der weiteren Argumente ersetzt:
  - %d Dezimalzahl
  - %x Hexadezimalzahl
  - %u Vorzeichenlose Zahl
  - %c ein Zeichen
  - %s Zeichenkette
  - %f Gleitkommazahl

Beispiel

```
printf( "1+2=%d\n", (1+2) );

int i=42;
printf( "%d als Hex.-Zahl: %x\n", i,i );

char *text = "Hallo";
printf( "%s\n", text );
```

scanf(Kontrollstring, arg1, arg2, ...)

- Gegenstück zu printf
- Liest Zeichen von der Standard-Eingabe, interpretiert sie gemäß des Kontrollstrings und speichert die Wert in den Argumenten
- **Achtung**: Argumente müssen Zeiger auf die Speicherbereiche sein, in denen die Werte abgelegt werden sollen.
- Kontrollstring enthält:
  - Whitespaces, die als Trennzeichen dienen und ignoriert werden
  - Gewöhnliche Zeichen, die als nächstes Zeichen in der Eingabe vorkommen müssen
  - Formatelement, die mit "%" beginnen

Beispiel

```
int i;
float x;
char str[80];
scanf("%d %f %s", &i, &x, str);
```

Eingabe von "25 23.12 Ralf" weist i den Wert 25 zu, x den Wert 23.12 und str wird mit Ralf und abschließender Null gefüllt.

Modularisierung

Um in C Funktionen anderer Module verwenden zu können, müssen diese Funktionen zunächst *deklariert* werden. Hierdurch wird dem Compiler mitgeteilt, welche Parameter eine Funktion erwartet und welchen Rückgabewert sie besitzt.

```
void *malloc( size_t size );
```

Der dazugehörige Programmcode ist in einer Objektdatei abgelegt, die zum Schluss zum Programm "hinzugelinkt" wird (siehe Compiler-Handbuch).

Header Dateien

Für ein Modul werden die Funktionsprototypen in der Regel in Header-Dateien (.h) zusammengefasst, die mittels

```
#include "dateiname.h"   bzw.   #include <dateiname.h>
```

eingebunden werden können.

Beispiele häufig verwendeter Header Dateien

stdio.h:	Funktionen zur Ein-/Ausgabe (printf, scanf, fopen, fclose, ...)
string.h:	Funktionen zur Manipulation von Zeichenketten (strlen, strcpy, ...)
math.h:	Mathematische Funktionen (sin, cos, sqrt, ...)

C-Specials

Die Operatoren ++ und --

- 2 Varianten: ++a und a++ (-- analog)
- Beides sind Abkürzungen für a=a+1
- Unterschied: Zeitpunkt der Inkrementierung
- Beispiel:
  i=0; a=7; i=++a; i und a haben den Wert 8
  i=0; a=7; i=a++; i hat den Wert 7, a den Wert 8

Die Operatoren +=, -=, *=, /=, &=, ...

a += 7;
ist eine Abkürzung für
a = a + 7;

?-Operator

x = (Bed ? Wert1 : Wert2);
ist eine Abkürzung für
if (Bed)
    x = Wert1;
else
    x = Wert2;

Beispiel:

```
int maxi ( int a, int b )
{
    return (a>b ? a : b);
}
```


C-Specials

Mehrfachzuweisungen

Falls einer Reihe von Variablen derselbe Wert zugewiesen werden soll, kann dies durch eine Anweisung geschehen:

a = b = c = d = e = 7;

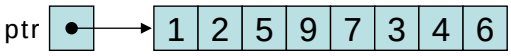
statt

a=7; b=7; c=7; d=7; e=7;

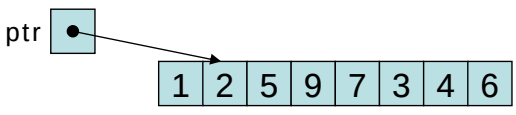
Rechnen mit Zeigern

In C kann in weitem Rahmen mit Zeigern gerechnet werden. Ist ptr ein Zeiger, setzt bspw. ptr++ den Zeiger auf das nächste Element. Der Comiler berücksichtigt dabei die Größe der Daten, auf die der Zeiger zeigt (z.B. 4 Byte für int, 1 Byte für char, ...).

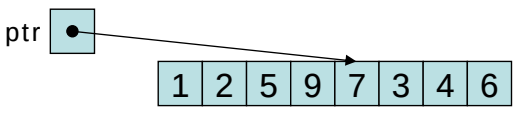
Beispiel




```
char *ptr;
ptr++;
```




```
int *ptr;
ptr++;
```



Beispiele

strcpy: Vom Anfänger zum Profi ...

```
void strcpy( char *dest, char *src )
{
      int i;
      for (i=0; src[i] != '\0'; i=i+1)
            dest[i] = src[i];
      dest[i] = '\0';
}
```

```
void strcpy( char *dest, char *src )
{
      while ( *src != '\0' )
      {
            *dest = *src;
            src++;
            dest++;
      }
      *dest = '\0';
}
```

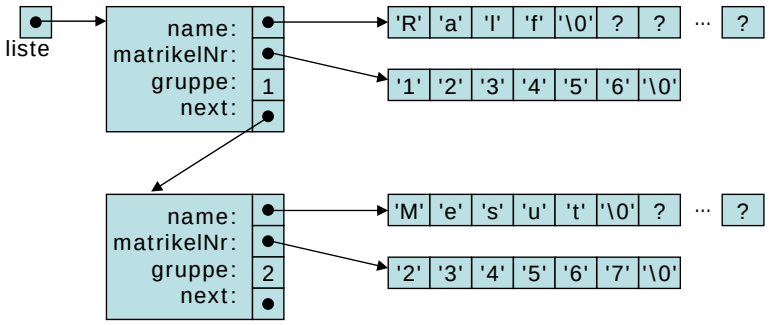
```
void strcpy( char *dest, char *src )
{
      int i;
      while( (dest[i] = src[i]) != '\0' )
            i = i+1;
}
```

```
void strcpy( char *dest, char *src )
{
      while ( *dest++ = *src++ );
}
```


Beispiele

Verkettete Liste

```
typedef struct _person {
      char *name;
      char matrikelNr[7];
      enum {Mi,Do,Fr} gruppe;
      struct _person *next;
} tPerson;

tPerson *liste = NULL;
tPerson *person = NULL;
```




```
liste = person = (tPerson *)malloc( sizeof( tPerson ));
person->name = (char *)malloc( 80 * sizeof( char ));
strcpy( person->name, "Ralf" );
strcpy( person->matrikelNr, "123456" );
person->gruppe = Do;
person->next = NULL;

liste->next = person = (tPerson *)malloc( sizeof( tPerson ));
person->name = (char *)malloc( 80 * sizeof( char ));
strcpy( person->name, "Mesut" );
strcpy( person->matrikelNr, "234567" );
person->gruppe = Fr;
person->next = NULL;
```